INTERNET-DRAFT
Common Authentication Technology WG
draft-ietf-cat-krb5gss-mech2-03.txt

Tom Yu
MIT

04 March 2000

The Kerberos Version 5 GSSAPI Mechanism, Version 2

Status of This Memo

Abstract

This document defines protocols, procedures, and conventions to be employed by peers implementing the Generic Security Service Application Program Interface (as specified in RFC 2743) when using Kerberos Version 5 technology (as specified in RFC 1510). This obsoletes RFC 1964.

Acknowledgements

Much of the material in this specification is based on work done for Cygnus Solutions by Marc Horowitz.

Table of Contents

1. Introduction

The original Kerberos 5 GSSAPI mechanism[RFC1964] has a number of shortcomings. This document attempts to remedy them by defining a completely new Kerberos 5 GSSAPI mechanism.

The context establishment token format requires that the authenticator of AP-REQ messages contain a cleartext data structure in its checksum field, which is a needless and potentially confusing overloading of that field. This is implemented by a special checksum algorithm whose purpose is to copy the input data directly into the checksum field of the authenticator.

The number assignments for checksum algorithms and for encryption types are inconsistent between the Kerberos protocol and the original GSSAPI mechanism. If new encryption or checksum algorithms are added to the Kerberos protocol at some point, the GSSAPI mechanism will need to be separately updated to use these new algorithms.

The original mechanism specifies a crude method of key derivation (by using the XOR of the context key with a fixed constant), which is incompatible with newer cryptosystems which specify key derivation procedures themselves. The original mechanism also assumes that both checksums and cryptosystem blocksizes are eight bytes.

Defining all GSSAPI tokens for the new Kerberos 5 mechanism in terms of the Kerberos protocol specification ensures that new encryption types and checksum types may be automatically used as they are defined for the Kerberos protocol.

2. Token Formats

All tokens, not just the initial token, are framed as the InitialContextToken described in RFC 2743 section 3.1. The innerContextToken element of the token will not itself be encoded in ASN.1, with the exception of caller-provided application data.

One rationale for avoiding the use of ASN.1 in the inner token is that some implementors may wish to implement this mechanism in a kernel or other similarly constrained application where handling of full ASN.1 encoding may be cumbersome. Also, due to the poor availability of the relevant standards documents, ASN.1 encoders and decoders are difficult to implement completely correctly, so keeping ASN.1 usage to a minimum decreases the probability of bugs in the implementation of the mechanism. In particular, bit strings need to be transferred at certain points in this mechanism. There are many conflicting common misunderstandings of how to encode and decode ASN.1 bit strings, which have led difficulties in the implementaion of the Kerberos protocol.

2.1. Packet Notation

The order of transmission of this protocol is described at the octet level. Packet diagrams depict bits in the order of transmission, assuming that individual octets are transmitted with the most significant bit (MSB) first. The diagrams read from left to right and from top to bottom, as in printed English. In each octet, bit number 7 is the MSB and bit number 0 is the LSB.

Numbers prefixed by the characters "0x" are in hexadecimal notation, as in the C programming language. Even though packet diagrams are drawn 16 bits wide, no padding should be used to align the ends of variable-length fields to a 32-bit or 16-bit boundary.

All integer fields are in network byte order. All other fields have the size shown in the diagrams, with the exception of variable length fields.

2.2. Mechanism OID

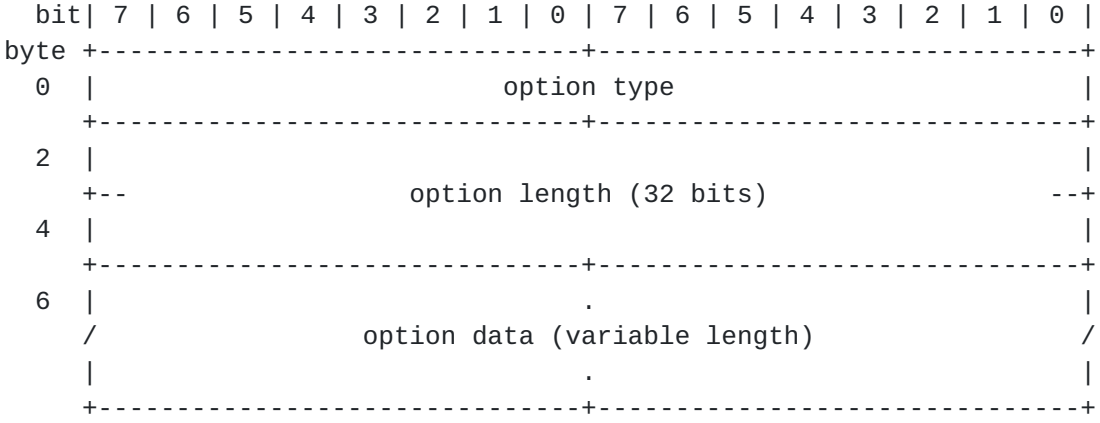
The Object Identifier (OID) of the new krb5 v2 mechanism is:

{iso(1) member-body(2) us(840) mit(113554) infosys(1) gssapi(2) krb5v2(3)}

2.3. Context Establishment

2.3.1. Option Format

Context establishment tokens, i.e., the initial ones that the GSS_Init_sec_context() and the GSS_Accept_sec_context() calls emit while a security context is being set up, may contain options that influence the subsequent behavior of the context. This document describes only a small set of options, but additional types may be added by documents intended to supplement this one. The generic format is as follows:



option type (16 bits)
The type identifier of the following option.

option length (32 bits)
The length in bytes of the following option.

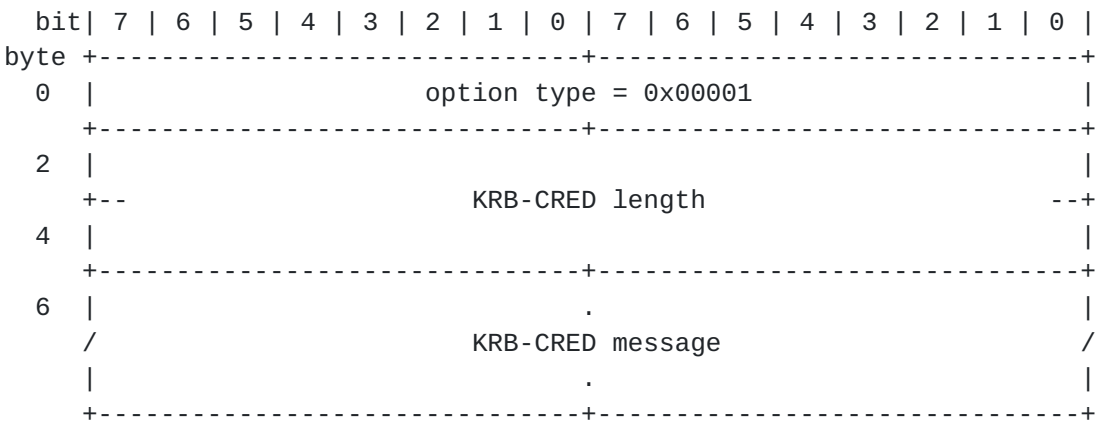
option data (variable length)
The actual option data.

Any number of options may appear in an initator or acceptor token. The final option in a token must be the null option, in order to mark the end of the list. Option type 0xffff is reserved.

The initiator and acceptor shall ignore any options that they do not understand.

2.3.1.1. Delegated Credentials Option

Only the initiator may use this option. The format of the delegated credentials option is as follows:



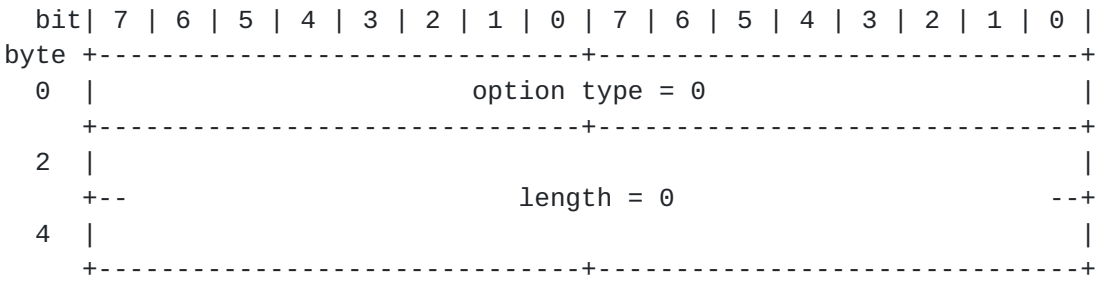
option type (16 bits)
The option type for this option shall be 0x0001.

KRB-CRED length (32 bits)
The length in bytes of the following KRB-CRED message.

KRB-CRED message (variable length)
The option data for this option shall be the KRB-CRED message that contains the credentials being delegated (forwarded) to the context acceptor. Only the initiator may use this option.

2.3.1.2. Null Option

The Null option terminates the option list, and must be used by both the initiator and the acceptor. Its format is as follows:

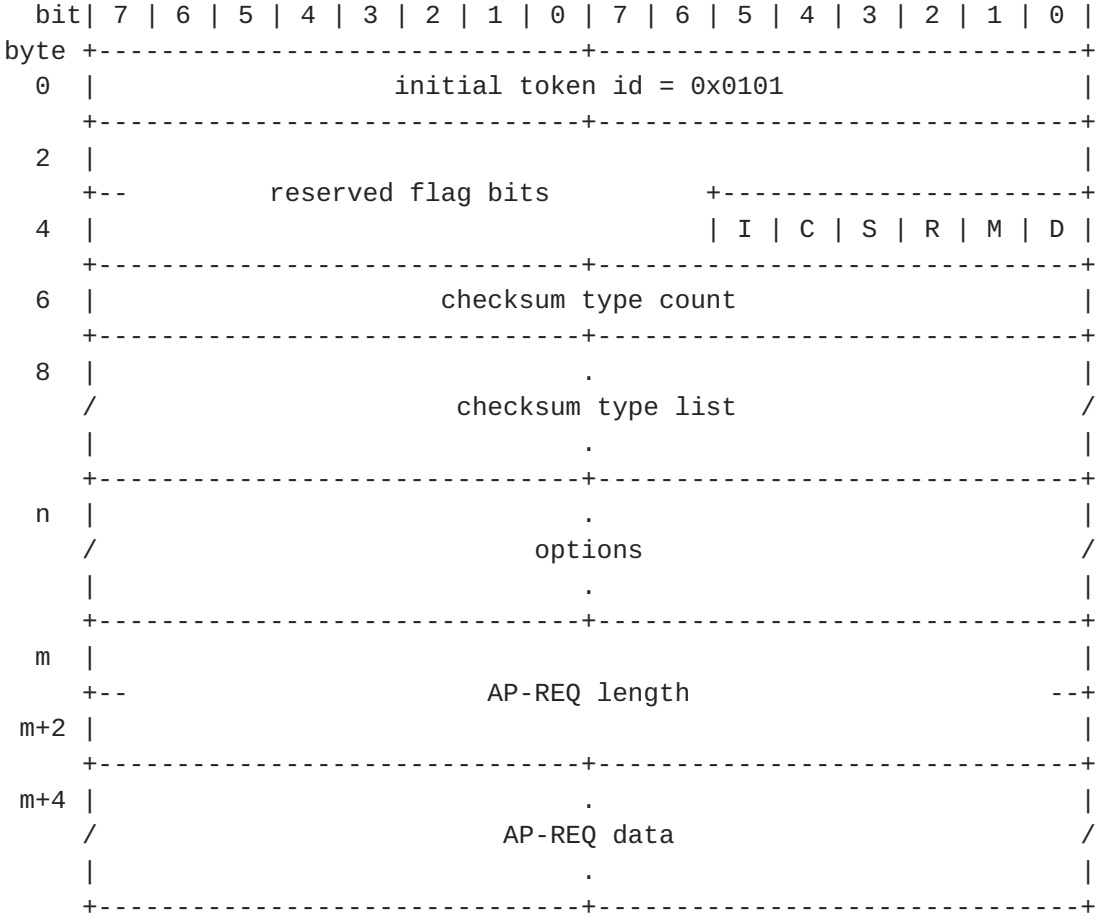


option type (16 bits)
The option type of this option must be zero.

option length (32 bits)
The length of this option must be zero.

2.3.2. Initial Token

This is the initial token sent by the context initiator, generated by GSS_Init_sec_context().



initial token ID (16 bits)

Contains the integer 0x0101, which identifies this as the initial token in the context setup.

reserved flag bits (26 bits)
These bits are reserved for future expansion. They must be set to zero by the initiator and be ignored by the acceptor.

I flag (1 bit)
0x00000020 -- GSS_C_INTEG_FLAG

C flag (1 bit)
0x00000010 -- GSS_C_CONF_FLAG

S flag (1 bit)
0x00000008 -- GSS_C_SEQUENCE_FLAG

R flag (1 bit)
0x00000004 -- GSS_C_REPLAY_FLAG

M flag (1 bit)
0x00000002 -- GSS_C_MUTUAL_FLAG

D flag (1 bit)
0x00000001 -- GSS_C_DELEG_FLAG; This flag must be set if the "delegated credentials" option is included.

checksum type count (16 bits)
The number of checksum types supported by the initiator.

checksum type list (variable length)
A list of Kerberos checksum types, as defined in RFC 1510 section 6.4. These checksum types must be collision-proof and keyed with the context key; no checksum types that are incompatible with the encryption key shall be used. Each checksum type number shall be 32 bits wide. This list should contain all the checksum types supported by the initiator. If mutual authentication is not used, then this list shall contain only one checksum type.

options (variable length)
The context initiation options, described in section 2.3.1.

AP-REQ length (32 bits)
The length of the following KRB_AP_REQ message.

AP-REQ data (variable length)
The AP-REQ message as described in RFC 1510. The checksum in the authenticator will be computed over the items listed in the next section.

The optional sequence number field shall be used in the AP-REQ. The initiator should generate a subkey in the authenticator, and the acceptor should generate a subkey in the AP-REP. The key used for

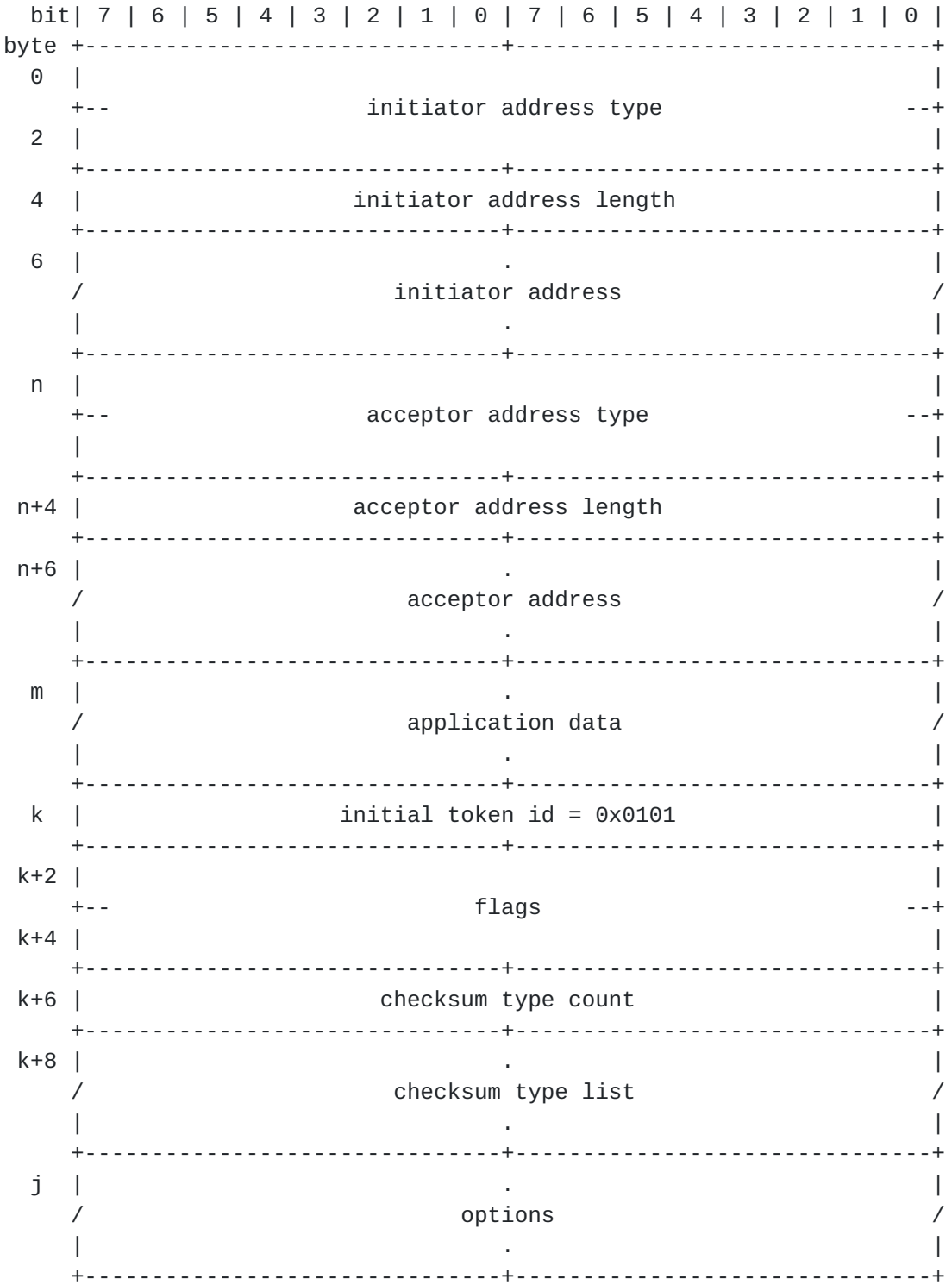
the per-message tokens will be the AP-REP subkey, or if that is not present, the authenticator subkey, or if that is not present, the session key. When subkeys are generated, it is strongly recommended

that they be of the same type as the associated session key.

XXX The above is not secure. There should be an algorithmic process to arrive at a subsession key which both sides of the authentication exchange can perform based on the ticket sessions key and data known to both parties, and this should probably be part of the revised Kerberos protocol rather than bound to the GSSAPI mechanism.

2.3.2.1. Data to be Checksummed in AP-REQ

The checksum in the AP-REQ message is calculated over the following items. Like in the actual tokens, no padding should be added to force integer fields to align on 32 bit boundaries. This particular set of data should not be sent as a part of any token; it merely specifies what is to be checksummed in the AP-REQ. The items in this encoding that precede the initial token ID correspond to the channel bindings passed to GSS_Init_sec_context().



initiator address type (32 bits)
: The initiator address type, as defined in the Kerberos protocol specification. If no initiator address is provided, this must be zero.

initiator address length (16 bits)
The length in bytes of the following initiator address. If there is no inititator address provided, this must be zero.

initiator address (variable length)
The actual initiator address, in network byte order.

acceptor address type (32 bits)
The acceptor address type, as defined in the Kerberos protocol specification. If no acceptor address is provided, this must be zero.

acceptor address length (16 bits)
The length in bytes of the following acceptor address. This must be zero is there is no acceptor address provided.

initiator address (variable length)
The actual acceptor address, in network byte order.

applicatation data (variable length)
The application data, if provided, encoded as a ASN.1 octet string using DER. If no application data are passed as input channel bindings, this shall be a zero-length ASN.1 octet string.

initial token ID (16 bits)
The initial token ID from the initial token.

flags (32 bits)
The context establishment flags from the initial token.

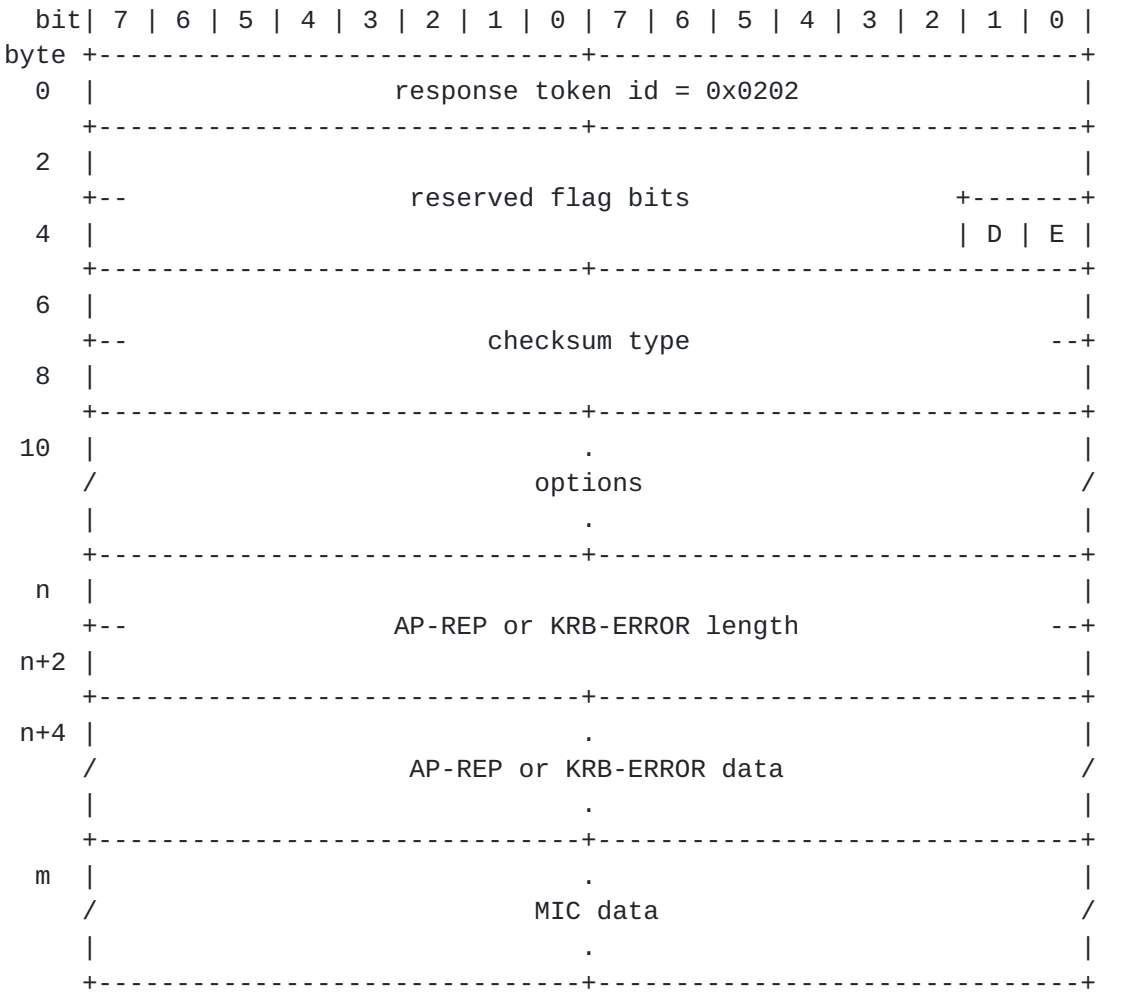
checksum type count (16 bits)
The number of checksum types supported by the initiator.

checksum type list (variable length)
The same list of checksum types contained in the initial token.

options (variable length)
The options list from the initial token.

2.3.3. Response Token

This is the reponse token sent by the context acceptor, if mutual authentication is enabled.



response token id (16 bits)
: Contains the integer 0x0202, which identifies this as the response token in the context setup.

reserved flag bits (30 bits)
: These bits are reserved for future expansion. They must be set to zero by the acceptor and be ignored by the initiator.

D flag -- delegated creds accepted (1 bit)
: 0x00000002 -- If this flag is set, the acceptor processed the delegated credentials, and GSS_C_DELEG_FLAG should be returned to the caller.

E flag -- error (1 bit)
: 0x00000001 -- If this flag is set, a KRB-ERROR message shall be present, rather than an AP-REP message. If this flag is not set, an AP-REP message shall be present.

checksum type count (16 bits)

The number of checksum types supported by both the initiator and the acceptor.

checksum type (32 bits)
: A Kerberos checksum type, as defined in RFC 1510 section 6.4. This checksum type must be among the types listed by the initiator, and will be used in for subsequent checksums generated during this security context.

options (variable length)
: The option list, as described earlier. At this time, no options are defined for the acceptor, but an implementation might make use of these options to acknowledge an option from the initial token. After all the options are specified, a null option must be used to terminate the list.

AP-REP or KRB-ERROR length (32 bits)
: Depending on the value of the error flag, length in bytes of the AP-REP or KRB-ERROR message.

AP-REP or KRB-ERROR data (variable length)
: Depending on the value of the error flag, the AP-REP or KRB-ERROR message as described in RFC 1510. If this field contains an AP-REP message, the sequence number field in the AP-REP shall be filled. If this is a KRB-ERROR message, no further fields will be in this message.

MIC data (variable length)
: A MIC token, as described in section 2.4.2, computed over the concatentation of the response token ID, flags, checksum length and type fields, and all option fields. This field and the preceding length field must not be present if the error flag is set.

2.4. Per-message Tokens

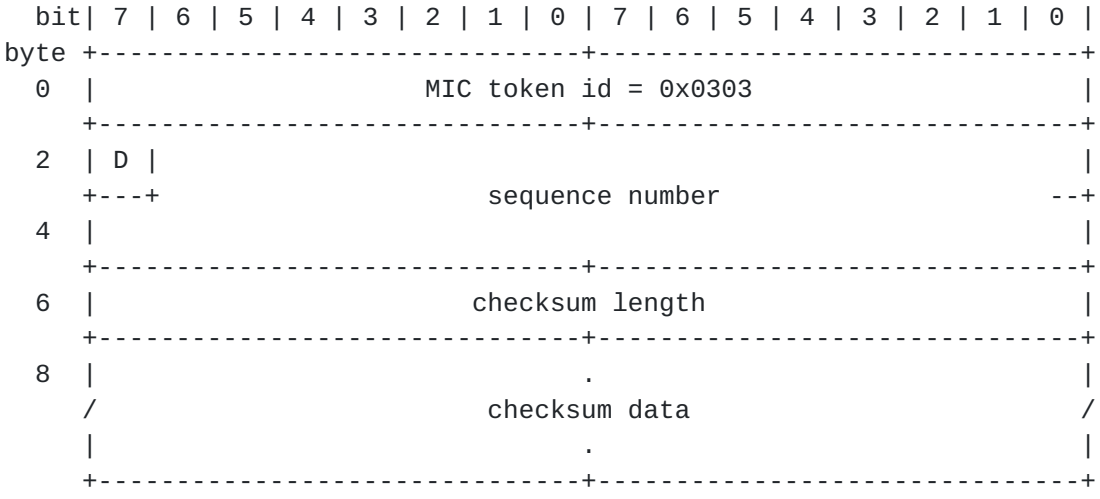
2.4.1. Sequence Number Usage

Sequence numbers for per-message tokens are 31 bit unsigned integers, which are incremented by 1 after each token. An overflow condition should result in a wraparound of the sequence number to zero. The initiator and acceptor each keep their own sequence numbers per connection.

The intial sequence number for tokens sent from the initiator to the acceptor shall be the least significant 31 bits of sequence number in the AP-REQ message. The initial sequence number for tokens sent from the acceptor to the initiator shall be the least significant 31 bits of the sequence number in the AP-REP message if mutual authentication is used; if mutual authentication is not used, the initial sequence number from acceptor to initiator shall be the least significant 31 bits of the sequence number in the AP-REQ message.

2.4.2. MIC Token

Use of the GSS_GetMIC() call yields a token, separate from the user data being protected, which can be used to verify the integrity of that data when it is received. The MIC token has the following format:



MIC token id (16 bits)
: Contains the integer 0x0303, which identifies this as a MIC token.

D -- direction bit (1 bit)
: This bit shall be zero if the message is sent from the context initiator. If the message is sent from the context acceptor, this bit shall be one.

sequence number (31 bits)
: The sequence number.

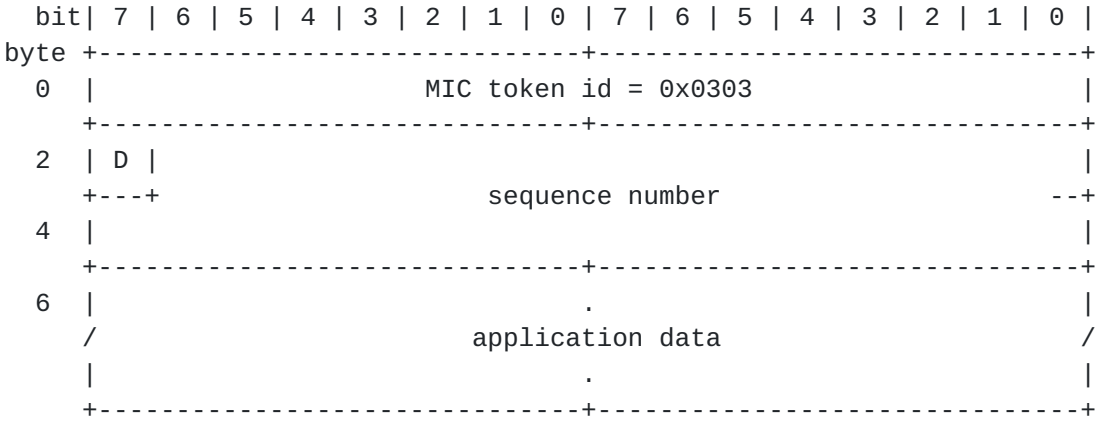
checksum length (16 bits)
: The number of bytes in the following checksum data field.

checksum data (variable length)
: The checksum itself, as defined in RFC 1510 section 6.4. The checksum is calculated over the encoding described in the following section. The key usage GSS_TOK_MIC -- 22 [XXX need to register this] shall be used in cryptosystems that support key derivation.

The mechanism implementation shall only use the checksum type returned by the acceptor in the case of mutual authentication. If mutual authentication is not requested, then only the checksum type in the initiator token shall be used.

2.4.2.1. Data to be Checksummed in MIC Token

The checksum in the MIC token shall be calculated over the following elements. This set of data is not actually included in the token as is; the description only appears for the purpose of specifying the method of calculating the checksum.



MIC token ID (16 bits)
The MIC token ID from the MIC message.

D -- direction bit (1 bit)
This bit shall be zero if the message is sent from the context initiator. If the message is sent from the context acceptor, this bit shall be one.

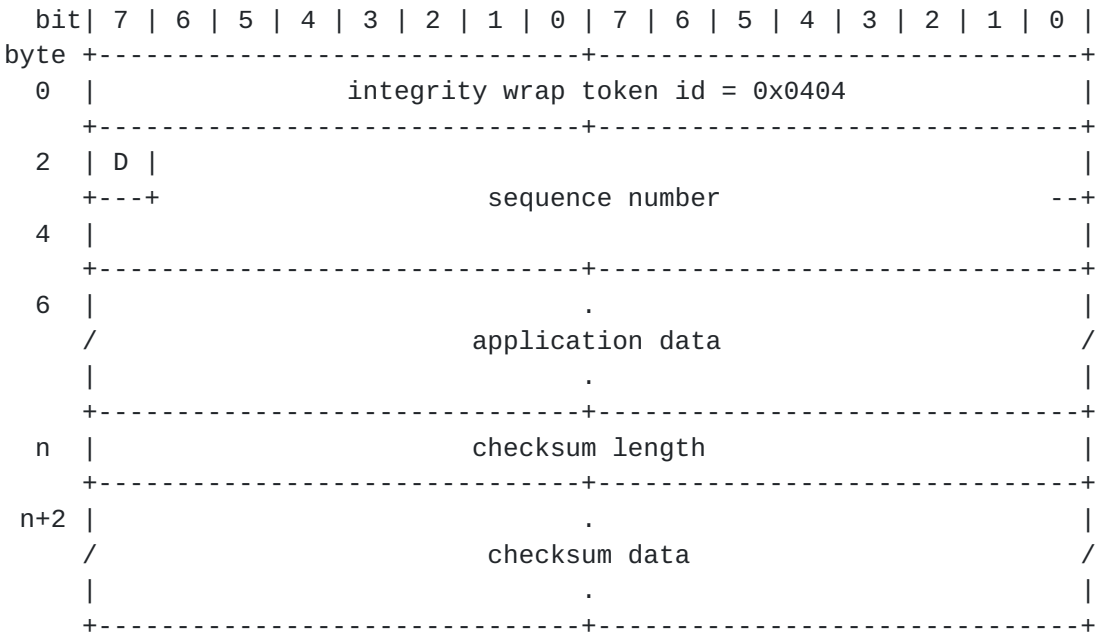
sequence number (31 bits)
The sequence number.

application data (variable length)
The application-supplied data, encoded as an ASN.1 octet string using DER.

2.4.3. Wrap Token

Use of the GSS_Wrap() call yields a token which encapsulates the input user data (optionally encrypted) along with associated integrity check quantities.

2.4.3.1. Wrap Token With Integrity Only



integrity wrap token id (16 bits)
: Contains the integer 0x0404, which identifies this as a Wrap token with integrity only.

D -- direction bit (1 bit)
: This bit shall be zero if the message is sent from the context initiator. If the message is sent from the context acceptor, this bit shall be one.

sequence number (31 bits)
: The sequence number.

application data (variable length)
: The application-supplied data, encoded as an ASN.1 octet string using DER.

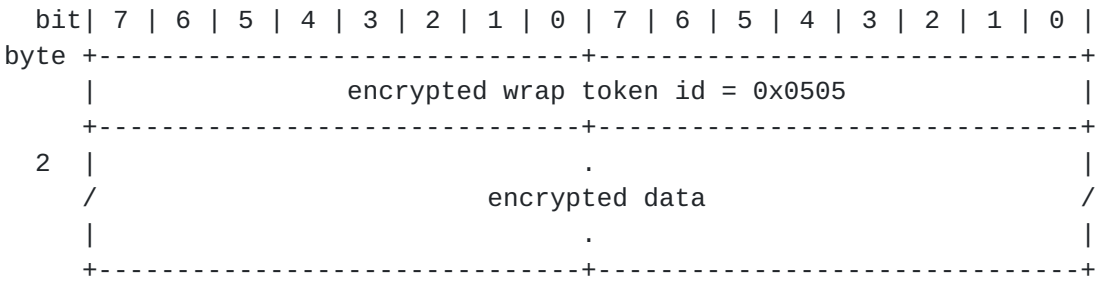
checksum length (16 bits)
: The number of bytes in the following checksum data field.

checksum data (variable length)
: The checksum itself, as defined in RFC 1510 section 6.4, computed over the concatenation of the token ID, sequence number, direction field, application data length, and application data, as in the MIC token checksum in the previous section. The key usage GSS_TOK_WRAP_INTEG -- 23 [XXX need to register this] shall be used in cryptosystems that support key derivation.

The mechanism implementation should only use checksum types which it

knows to be valid for both peers, as described for MIC tokens.

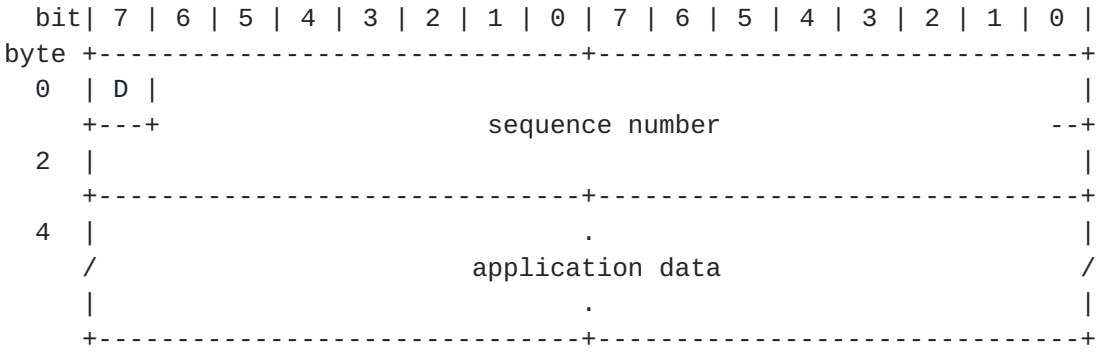
2.4.3.2. Wrap Token With Integrity and Encryption



encrypted wrap token id (16 bits)
: Contains the integer 0x0505, which identifies this as a Wrap token with integrity and encryption.

encrypted data (variable length)
: The encrypted data itself, as defined in RFC 1510 section 6.3, encoded as an ASN.1 octet string using DER. Note that this is not the ASN.1 type EncryptedData as defined in RFC 1510 section 6.1, but rather the ciphertext without encryption type or kvno information. The encryption is performed using the key/enctype exchanged during context setup. The confounder and checksum are as specified in the Kerberos protocol specification. The key usage GSS_TOK_WRAP_PRIV -- 24 [XXX need to register this] shall be used in cryptosystems that support key derivation. The actual data to be encrypted are specified below.

2.4.3.2.1. Data to be Encrypted in Wrap Token



D -- direction bit (1 bit)
: This bit shall be zero if the message is sent from the context initiator. If the message is sent from the context acceptor, this bit shall be one.

sequence number (31 bits)
: The sequence number.

application data (variable length)
The application-supplied data, encoded as an ASN.1 octet string

using DER.

3. ASN.1 Encoding of Octet Strings

In order to encode arbitirarly-sized application data, ASN.1 octet string encoding is in this protocol. The Distinguished Encoding Rules (DER) shall always be used in such cases. For reference purposes, the DER encoding of an ASN.1 octet string, adapted from ITU-T X.690, follows:

```
+--------+-------//-------+-------//-------+
|00000100| length octets  |contents octets |
+--------+-------//-------+-------//-------+
 |
 +-- identifier octet = 0x04 = [UNIVERSAL 4]
```

In this section only, the bits in each octet shall be numbered as in the ASN.1 specification, from 8 to 1, with bit 8 being the MSB of the octet, and with bit 1 being the LSB of the octet.

identifier octet (8 bits)
: Contains the constant 0x04, the tag for primitive encoding of an octet string with the default (UNIVERSAL 4) tag.

length octets (variable length)
: Contains the length of the contents octets, in definite form (since this encoding uses DER).

contents octets (variable length)
: The contents of the octet string.

The length octets shall consist of either a short form (one byte only), which is to be used only if the number of octets in the contents octets is less than or equal to 127, or a long form, which is to be used in all other cases. The short form shall consist of a single octet with bit 8 (the MSB) equal to zero, and the remaining bits encoding the number of contents octets (which may be zero) as an unsigned binary integer.

The long form shall consist of an initial octet and one or more subsequent octets. The first octet shall have bit 8 (the MSB) set to one, and the remaining bits shall encode the number of subsequent octets in the length encoding as an unsigned binary integer. The length must be encoded in the minimum number of octets. An initial octet of 0xFF is reserved by the ASN.1 specification. Bits 8 to 1 of the first subsequent octet, followed by bits 8 to 1 of each subsequent octet in order, shall be the encoding of an unsigned binary integer, with bit 8 of the first octet being the most significant bit. Thus, the length encoding within is in network byte

order.

An initial length octet of 0x80 shall not be used, as that is reserved by the ASN.1 specification for indefinite lengths in conjunction with constructed contents encodings, which are not to be used with DER.

4. Name Types

This section discusses the name types which may be passed as input to the Kerberos 5 GSSAPI mechanism's GSS_Import_name() call, and their associated identifier values. It defines interface elements in support of portability, and assumes use of C language bindings per RFC 2744. In addition to specifying OID values for name type identifiers, symbolic names are included and recommended to GSSAPI implementors in the interests of convenience to callers. It is understood that not all implementations of the Kerberos 5 GSSAPI mechanism need support all name types in this list, and that additional name forms will likely be added to this list over time. Further, the definitions of some or all name types may later migrate to other, mechanism-independent, specifications. The occurrence of a name type in this specification is specifically not intended to suggest that the type may be supported only by an implementation of the Kerberos 5 mechanism. In particular, the occurrence of the string "_KRB5_" in the symbolic name strings constitutes a means to unambiguously register the name strings, avoiding collision with other documents; it is not meant to limit the name types' usage or applicability.

For purposes of clarification to GSSAPI implementors, this section's discussion of some name forms describes means through which those forms can be supported with existing Kerberos technology. These discussions are not intended to preclude alternative implementation strategies for support of the name forms within Kerberos mechanisms or mechanisms based on other technologies. To enhance application portability, implementors of mechanisms are encouraged to support name forms as defined in this section, even if their mechanisms are independent of Kerberos 5.

4.1. Mandatory Name Forms

This section discusses name forms which are to be supported by all conformant implementations of the Kerberos 5 GSSAPI mechanism.

4.1.1. Kerberos Principal Name Form

This name form shall be represented by the Object Identifier {iso(1) member-body(2) us(840) mit(113554) infosys(1) gssapi(2) krb5(2) krb5_name(1)}. The recommended symbolic name for this type is "GSS_KRB5_NT_PRINCIPAL_NAME".

This name type corresponds to the single-string representation of a Kerberos name. (Within the MIT Kerberos 5 implementation, such names are parseable with the krb5_parse_name() function.) The elements included within this name representation are as follows, proceeding

from the beginning of the string:

(1) One or more principal name components; if more than one principal name component is included, the components are separated by '/'. Arbitrary octets may be included within principal name components, with the following constraints and special considerations:

(1a) Any occurrence of the characters '@' or '/' within a name component must be immediately preceded by the '\' quoting character, to prevent interpretation as a component or realm separator.

(1b) The ASCII newline, tab, backspace, and null characters may occur directly within the component or may be represented, respectively, by '\n', '\t', '\b', or '\0'.

(1c) If the '\' quoting character occurs outside the contexts described in (1a) and (1b) above, the following character is interpreted literally. As a special case, this allows the doubled representation '\\' to represent a single occurrence of the quoting character.

(1d) An occurrence of the '\' quoting character as the last character of a component is illegal.

(2) Optionally, a '@' character, signifying that a realm name immediately follows. If no realm name element is included, the local realm name is assumed. The '/' , ':', and null characters may not occur within a realm name; the '@', newline, tab, and backspace characters may be included using the quoting conventions described in (1a), (1b), and (1c) above.

4.1.2. Exported Name Object Form for Kerberos 5 Mechanism

When generated by the Kerberos 5 mechanism, the Mechanism OID within the exportable name shall be that of the original Kerberos 5 mechanism[RFC1964]. The Mechanism OID for the original Kerberos 5 mechanism is:

{iso(1) member-body(2) us(840) mit(113554) infosys(1) gssapi(2) krb5(2)}

The name component within the exportable name shall be a contiguous string with structure as defined for the Kerberos Principal Name Form.

In order to achieve a distinguished encoding for comparison purposes, the following additional constraints are imposed on the export operation:

(1) all occurrences of the characters '@', '/', and '\' within principal components or realm names shall be quoted with an

immediately-preceding '\'.

(2) all occurrences of the null, backspace, tab, or newline characters within principal components or realm names will be represented, respectively, with '\0', '\b', '\t', or '\n'.

(3) the '\' quoting character shall not be emitted within an exported name except to accomodate cases (1) and (2).

5. Credentials

The Kerberos 5 protocol uses different credentials (in the GSSAPI sense) for initiating and accepting security contexts. Normal clients receive a ticket-granting ticket (TGT) and an associated session key at "login" time; the pair of a TGT and its corresponding session key forms a credential which is suitable for initiating security contexts. A ticket-granting ticket, its session key, and any other (ticket, key) pairs obtained through use of the ticket-granting-ticket, are typically stored in a Kerberos 5 credentials cache, sometimes known as a ticket file.

The encryption key used by the Kerberos server to seal tickets for a particular application service forms the credentials suitable for accepting security contexts. These service keys are typically stored in a Kerberos 5 key table (keytab), or srvtab file (the Kerberos 4 terminology). In addition to their use as accepting credentials, these service keys may also be used to obtain initiating credentials for their service principal.

The Kerberos 5 mechanism's credential handle may contain references to either or both types of credentials. It is a local matter how the Kerberos 5 mechanism implementation finds the appropriate Kerberos 5 credentials cache or key table.

However, when the Kerberos 5 mechanism attempts to obtain initiating credentials for a service principal which are not available in a credentials cache, and the key for that service principal is available in a Kerberos 5 key table, the mechanism should use the service key to obtain initiating credentials for that service. This should be accomplished by requesting a ticket-granting-ticket from the Kerberos Key Distribution Center (KDC), and decrypting the KDC's reply using the service key.

6. Parameter Definitions

This section defines parameter values used by the Kerberos V5 GSSAPI mechanism. It defines interface elements in support of portability, and assumes use of C language bindings per RFC 2744.

6.1. Minor Status Codes

This section recommends common symbolic names for minor_status values to be returned by the Kerberos 5 GSSAPI mechanism. Use of these

definitions will enable independent implementors to enhance application portability across different implementations of the mechanism defined in this specification. (In all cases, implementations of GSS_Display_status() will enable callers to convert minor_status indicators to text representations.) Each implementation should make available, through include files or other means, a facility to translate these symbolic names into the concrete values which a particular GSSAPI implementation uses to represent the minor_status values specified in this section.

It is recognized that this list may grow over time, and that the need for additional minor_status codes specific to particular implementations may arise. It is recommended, however, that implementations should return a minor_status value as defined on a mechanism-wide basis within this section when that code is accurately representative of reportable status rather than using a separate, implementation-defined code.

6.1.1. Non-Kerberos-specific codes

These symbols should likely be incorporated into the generic GSSAPI C-bindings document, since they really are more general.

```
GSS_KRB5_S_G_BAD_SERVICE_NAME
        /* "No @ in SERVICE-NAME name string" */
GSS_KRB5_S_G_BAD_STRING_UID
        /* "STRING-UID-NAME contains nondigits" */
GSS_KRB5_S_G_NOUSER
        /* "UID does not resolve to username" */
GSS_KRB5_S_G_VALIDATE_FAILED
        /* "Validation error" */
GSS_KRB5_S_G_BUFFER_ALLOC
        /* "Couldn't allocate gss_buffer_t data" */
GSS_KRB5_S_G_BAD_MSG_CTX
        /* "Message context invalid" */
GSS_KRB5_S_G_WRONG_SIZE
        /* "Buffer is the wrong size" */
GSS_KRB5_S_G_BAD_USAGE
        /* "Credential usage type is unknown" */
GSS_KRB5_S_G_UNKNOWN_QOP
        /* "Unknown quality of protection specified" */
```

6.1.2. Kerberos-specific-codes

```
GSS_KRB5_S_KG_CCACHE_NOMATCH
        /* "Principal in credential cache does not match desired name" */
GSS_KRB5_S_KG_KEYTAB_NOMATCH
        /* "No principal in keytab matches desired name" */
GSS_KRB5_S_KG_TGT_MISSING
        /* "Credential cache has no TGT" */
GSS_KRB5_S_KG_NO_SUBKEY
        /* "Authenticator has no subkey" */
GSS_KRB5_S_KG_CONTEXT_ESTABLISHED
        /* "Context is already fully established" */
GSS_KRB5_S_KG_BAD_SIGN_TYPE
        /* "Unknown signature type in token" */
GSS_KRB5_S_KG_BAD_LENGTH
        /* "Invalid field length in token" */
GSS_KRB5_S_KG_CTX_INCOMPLETE
        /* "Attempt to use incomplete security context" */
```

7. Kerberos Protocol Dependencies

This protocol makes several assumptions about the Kerberos protocol, which may require changes to the successor of RFC 1510.

Sequence numbers, checksum types, and address types are assumed to be no wider than 32 bits. The Kerberos protocol specification might need to be modified to accomodate this. This obviously requires some further discussion.

Key usages need to be registered within the Kerberos protocol for use with GSSAPI per-message tokens. The current specification of the Kerberos protocol does not include descriptions of key derivations or key usages, but planned revisions to the protocol will include them.

This protocol also makes the assumption that any cryptosystem used with the session key will include integrity protection, i.e., it assumes that no "raw" cryptosystems will be used.

8. Security Considerations

The GSSAPI is a security protocol; therefore, security considerations are discussed throughout this document. The original Kerberos 5 GSSAPI mechanism's constraints on possible cryptosystems and checksum types do not permit it to be readily extended to accomodate more secure cryptographic technologies with larger checksums or encryption block sizes. Sites are strongly encouraged to adopt the mechanism specified in this document in the light of recent publicity about the deficiencies of DES.

9. References

[X.680] ISO/IEC, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T X.680 (1997) | ISO/IEC 8824-1:1998

[X.690] ISO/IEC, "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T X.690 (1997) | ISO/IEC 8825-1:1998.

[RFC1510] Kohl, J., Neumann, C., "The Kerberos Network Authentication Service (V5)", RFC 1510.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface, Version 2, Update 1", RFC 2743.

[RFC2744] Wray, J., "Generic Security Service API Version 2: C-bindings", RFC 2744.

10. Author's Address

Tom Yu
Massachusetts Institute of Technology
Room E40-345
77 Massachusetts Avenue
Cambridge, MA 02139
USA

email: tlyu@mit.edu
phone: +1 617 253 1753